

“COMPRESS AND ELIMINATE” SOLVER FOR SYMMETRIC POSITIVE DEFINITE SPARSE MATRICES[§]

DARIA A. SUSHNIKOVA[¶] AND IVAN V. OSELEDETS^{‡¶}

Abstract. We propose a new approximate factorization for solving linear systems with symmetric positive definite sparse matrices. In a nutshell the algorithm is to apply hierarchically block Gaussian elimination and additionally compress the fill-in. The systems that have efficient compression of the fill-in mostly arise from discretization of partial differential equations. We show that the resulting factorization can be used as an efficient preconditioner and compare the proposed approach with the state-of-art direct and iterative solvers.

Key words.

Sparse matrix, direct solver, hierarchical matrix, symmetric positive-definite matrix

AMS subject classifications. 65F05, 65F50, 65F08

1. Introduction. We consider a linear system with a large sparse symmetric positive definite (SPD) matrix $A = A^\top > 0$, $A \in \mathbb{R}^{n \times n}$,

$$Ax = b.$$

Such systems typically come from the discretization of partial differential equations. It is well known that sparse Cholesky factorization has $\mathcal{O}(n)$ complexity only for very simple problems (for example, discretization of one-dimensional PDEs). As an alternative, hierarchical low-rank approximations have been used to approximate big dense blocks in the Cholesky factors. These methods can achieve optimal complexity for large classes of sparse linear systems. However the constant in $\mathcal{O}(n)$ can be quite large, thus for practically interesting values of n they often lose to “purely sparse” software (CHOLMOD, PARDISO, UMFPACK).

In this paper we try to combine those approaches and propose a new algorithm for (approximate) Cholesky-type factorization of sparse matrices. We start the standard block Gaussian elimination, but after each step we apply row and column transformation to the new fill-ins in the so-called “far zone” to create new zeros. In this scheme we have discovered experimentally that it is possible to eliminate approximately half of the unknowns without introducing new non-zeros. The approach is then applied in the multilevel fashion. The most time-consuming part of the method is low-rank approximation of small matrices, which has to be done after each elimination step. Then we show that the resulting factorization can be used as an efficient preconditioner, and compare the efficiency with the state-of-the-art fast direct sparse solver CHOLMOD on a model example.

2. Algorithm. Before running the algorithm we select the permutation P and work with the matrix $\tilde{A} = PAP^\top$. This permutation is crucial for the algorithm. The choice of the permutation, its properties and its influence on the proposed algorithm is discussed in Section 2.7. The permutation P and the block size B define the division of the matrix A into sub-blocks. The block size B is the parameter of the algorithm

[‡]Skolkovo Institute of Science and Technology, Nobel St. 3, Skolkovo Innovation Center, Moscow, 143025 Moscow Region, Russia (i.oseledets@skolkovotech.ru)

[¶]Institute of Numerical Mathematics Russian Academy of Sciences, Gubkina St. 8, 119333 Moscow, Russia

[§]This work was supported by Russian Science Foundation grant 14-1100659

and is typically small, the selection of the block size is discussed later. For simplicity we use the same letter A to refer to the permuted matrix.

We represent the matrix A in the *compressed sparse block* (CSB) [31] format. The nonzero blocks are referred to as “close” blocks. The nonzero blocks that arise during the elimination are called “far” blocks¹. The first step is to eliminate the first block row.

2.1. Elimination of the first block row. Consider the matrix A in the following block form:

$$A = \begin{bmatrix} D_1 & A_{1r} \\ A_{1r}^\top & A_{1*} \end{bmatrix},$$

where D_1 is $B \times B$. The first block row \tilde{R}_1 can be written as

$$\tilde{R}_1 = \begin{bmatrix} D_1 & A_{1r} \end{bmatrix} = \begin{bmatrix} D_1 & C_1 & 0 \end{bmatrix} P_1^{\text{col}},$$

where C_1 corresponds to all close blocks. The matrix P_1^{col} is a permutation matrix. To get the first step of the block Cholesky factorization we factorize the diagonal block:

$$D_1 = \hat{L}_1 \hat{L}_1^\top.$$

Then we apply the block Cholesky factorization [11]:

$$A = \begin{bmatrix} \hat{L}_1 & 0 \\ A_{1r}^\top \hat{L}_1^{-1} & 0 \end{bmatrix} \begin{bmatrix} \hat{L}_1^\top & \hat{L}_1^{-\top} A_{1r} \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \tilde{A}_1^* \end{bmatrix} = \tilde{L}_1 \tilde{L}_1^\top + \tilde{A}_1, \quad (2.1)$$

where the block $\tilde{A}_1^* = A_{1*} - A_{1r}^\top D_1^{-1} A_{1r}$ is the Schur complement. The matrix \tilde{A}_1^* has the sparsity pattern equal to the union of sparsity patterns of the matrix A_{1*} and the fill-in matrix

$$\tilde{A}_{1F} = A_{1r}^\top D_1^{-1} A_{1r}. \quad (2.2)$$

New non-zero blocks can appear in positions (i, j) , where $i \in \mathcal{I}_1, j \in \mathcal{I}_1$, and \mathcal{I}_1 is the set of indices of nonzero blocks in the first block row \tilde{R}_1 , see Figure 2.1b.

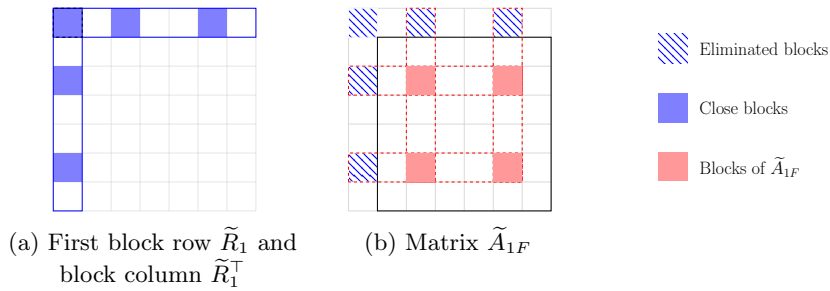


Fig. 2.1: Illustration of the fill-in caused by the first block row elimination

¹ “Far” blocks are typically called “fill” blocks, but we use “close” and “far” notation to emphasize the connection with the FMM method

Note that the number of non-zero blocks in the new matrix \tilde{A}_{1F} is bounded by $(\#\mathcal{I}_1)^2$. The positions of those blocks are known in advance. If the elimination is continued, the number of far blocks may grow. To avoid this problem we use an additional *compression procedure*.

2.2. Compression and elimination of i -th block row. Assume that we have already eliminated $(i-1)$ block rows and now we are working with the i -th block row, see Figure 2.2.

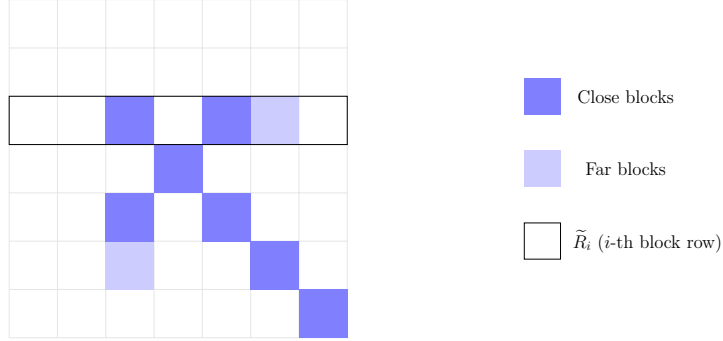


Fig. 2.2: Matrix \tilde{A}_{i-1} , starting point for general elimination ($i = 3$)

Here we consider the i -th block row of the matrix \tilde{A}_{i-1} and try to eliminate it. After all previous steps we obtain some far blocks, see Figure 2.2. We can reorder columns in the block row,

$$\tilde{R}_i = \begin{bmatrix} D_i & C_i & F_i & 0 \end{bmatrix} P_i^{\text{col}},$$

where P_i^{col} is the permutation matrix, $C_i = [C_i^{(1)} \dots C_i^{(n_i^C)}]$ are close blocks in the block row \tilde{R}_i , n_i^C is the number of close blocks in \tilde{R}_i , $F_i = [F_i^{(1)} \dots F_i^{(n_i^F)}]$ are far blocks in the block row \tilde{R}_i , n_i^F is the number of far blocks in \tilde{R}_i , see Figure 2.3.

The key idea is to approximate the matrix $F_i \in \mathbb{R}^{B \times (Bn_i^F)}$ by a low-rank matrix:

$$\tilde{U}_i^\top F_i = \begin{bmatrix} \hat{F}_i \\ E_i \end{bmatrix}, \quad (2.3)$$

where \tilde{U}_i is a unitary matrix, $\|E_i\| < \varepsilon$ for some ε and $\hat{F}_i \in \mathbb{R}^{r \times Bn_i^F}$.

REMARK 2.1. *Low-rank approximation presented in equation (2.2) is a crucial part of the proposed algorithm. We consider two strategies of finding the rank r :*

- $CE(\varepsilon)$: for a given accuracy ε find the minimum rank r such that $\|E_i\| < \varepsilon$ (adaptive low-rank approximation).
- $CE(r)$: Fix the rank r , typically $r = \frac{B}{2}$ (fixed rank approximation).

$CE(\varepsilon)$ algorithm leads to a better factorization accuracy and thus can be used as an approximate direct solver. A major drawback of the $CE(\varepsilon)$ algorithm is that the

compression of the far blocks is not guaranteed, which may lead to a memory-intensive factorization.

While $CE(r)$ algorithm directly controls the memory usage, it lacks control over the factorization accuracy. Therefore, $CE(r)$ is unsuitable for direct approximate solution of the system, but can be a good preconditioner.

Let

$$\tilde{Q}_i = \begin{bmatrix} I_{(i-1)B} & 0 & 0 \\ 0 & \tilde{U}_i & 0 \\ 0 & 0 & I_{(M-i)B} \end{bmatrix}, \quad (2.4)$$

then the matrix

$$\check{A}_{i-1} \approx \tilde{Q}_i^\top \check{A}_{i-1} \tilde{Q}_i$$

has the i -th row

$$\tilde{R}_{i*} = \begin{bmatrix} \tilde{U}_i^\top D_i \tilde{U}_i & \tilde{U}_i^\top C_i & \begin{bmatrix} \hat{F}_i \\ 0 \end{bmatrix} & 0 \end{bmatrix} P_i^{\text{col}},$$

see illustration on Figure 2.3.

The crucial part of the algorithm is that we eliminate only a part of the i -th block row of the modified matrix \check{A}_{i-1} , which has zero elements in the far zone. Introduce the block subrow $\tilde{S}_i \in \mathbb{R}^{(B-r) \times n}$ (red-dashed on Figure 2.3):

$$\tilde{S}_i = \begin{bmatrix} 0_{(B-r) \times r} & I_{(B-r)} \end{bmatrix} \tilde{R}_{i*}.$$

Now we can eliminate the block subrow from the matrix \check{A}_{i-1} using the block Cholesky elimination.

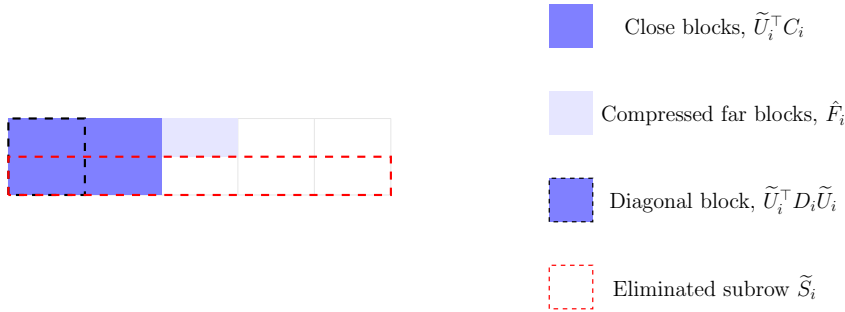


Fig. 2.3: Compression step

Denote the diagonal block of the block subrow \tilde{S}_i as \tilde{S}_{ii} . If the Cholesky decomposition of the diagonal block is $\tilde{S}_{ii} = \tilde{L}_i \hat{L}_i^\top$, then

$$\check{A}_{i-1} = \tilde{L}_i \tilde{L}_i^\top + \tilde{A}_i,$$

where the matrix $\tilde{A}_i \in \mathbb{R}^{n \times n}$ has zeros instead of subrow \tilde{S}_i and subcolumn \tilde{S}_i^\top , and

$$\tilde{L}_i = \tilde{S}_i^\top \hat{L}_i^{-\top}. \quad (2.5)$$

Note that in this scheme the block sparsity pattern of the block subrow \tilde{S}_i coincides with the block sparsity pattern of the original matrix A , thus after one elimination in the matrix \tilde{A}_i only blocks from the block sparsity pattern of the matrix $\tilde{A}_i^\top \tilde{A}_i$ can arise. Also note that the compression step does not affect locations and sizes of the far blocks, see Figure 2.4b. There is an important interpretation of the far blocks: far blocks sparsity pattern consists of blocks that are nonzero in the matrix A^2 and zero in the matrix A .

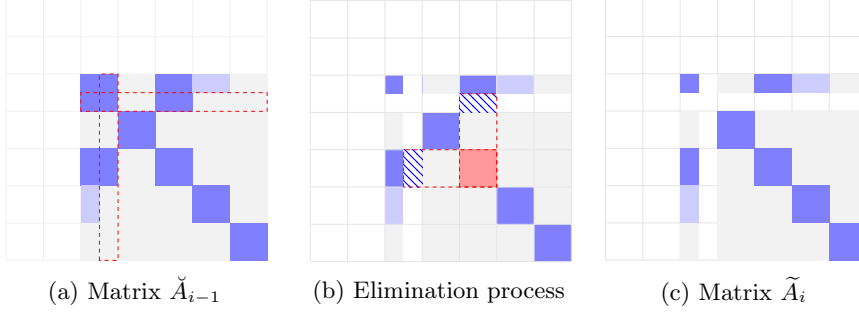


Fig. 2.4: Elimination step. Gray color denotes nonzero rows and columns of matrix \tilde{A}_i .

After the compression-elimination step, the matrix \tilde{A}_{i-1} is approximated as

$$\tilde{Q}_i \tilde{A}_{i-1} \tilde{Q}_i^\top \approx \tilde{L}_i \tilde{L}_i^\top + \tilde{A}_i,$$

where $\tilde{L}_i \in \mathbb{R}^{n \times (B-r)}$ and the matrix $\tilde{A}_i \in \mathbb{R}^{n \times n}$ have the same size as the initial matrix A .

2.3. Full sweep of the algorithm. Consider the compression-elimination procedure after all block rows have been processed. We store factors \tilde{L}_i multiplied by corresponding matrices \tilde{Q}_i as columns of the matrix \check{L}_1 :

$$\check{L}_1 = \left[\left(\prod_{j=M}^2 \tilde{Q}_j \right) \tilde{L}_1 \quad \cdots \quad \left(\prod_{j=M}^{i+1} \tilde{Q}_j \right) \tilde{L}_i \quad \cdots \quad \tilde{L}_M \right]. \quad (2.6)$$

We obtain the following approximation:

$$\tilde{Q}_M \cdots \tilde{Q}_1 A \tilde{Q}_1^\top \cdots \tilde{Q}_M^\top \approx \check{L}_1 \check{L}_1^\top + \hat{A}_1,$$

where the matrix $\check{L}_1 \in \mathbb{R}^{n \times n_{L1}}$, see Figure 2.5a, the matrix $\hat{A}_1 \in \mathbb{R}^{n \times n}$, see Figure 2.5c. Since multiplication by the matrix \tilde{Q}_i does not change the sparsity patterns of the matrices \check{L}_1 and \hat{A}_1 during the elimination, the block sparsity pattern of \check{L}_1 can be easily obtained from the block sparsity pattern of the original matrix. Denote

$$Q_1 = \prod_{i=M}^1 \tilde{Q}_i.$$

Due to a special structure of \tilde{Q}_i (2.14),

$$Q_1 = \text{diag}(\tilde{U}_1, \dots, \tilde{U}_M),$$

i.e. Q_1 is a block-diagonal matrix.

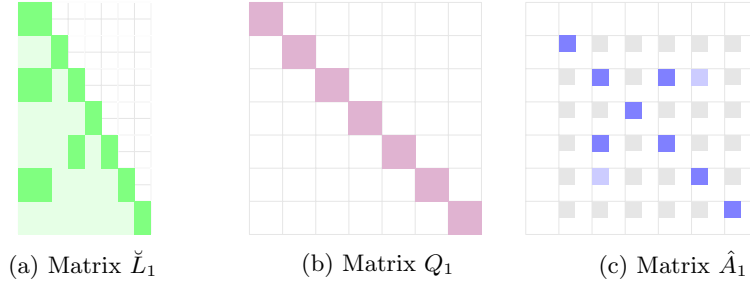


Fig. 2.5: Illustration of matrices \check{L}_1 , Q_1 and \hat{A}_1

Let us now permute the block rows of the matrix A in such a way that the eliminated rows are before the non-eliminated ones:

$$P_1 Q_1 A Q_1^\top P_1^\top \approx P_1 (\check{L}_1 \check{L}_1^\top + \hat{A}_1) P_1^\top = L_1 L_1^\top + \begin{bmatrix} 0_{n_{L_1} \times n_{L_1}} & 0 \\ 0 & A_1 \end{bmatrix}, \quad (2.7)$$

where

$$L_1 = P_1 \check{L}_1.$$

The matrix $L_1 \in \mathbb{R}^{n \times n_{L_1}}$ is a sparse block lower triangular matrix with the block size $(B - r) \times (B - r)$, $A_1 \in \mathbb{R}^{n_{A_1} \times n_{A_1}}$ is a block-sparse matrix $A_1 = P_1 \hat{A}_1 P_1^\top$ with the block size $r \times r$ and $n = n_{L_1} + n_{A_1}$, see Figure 2.6.

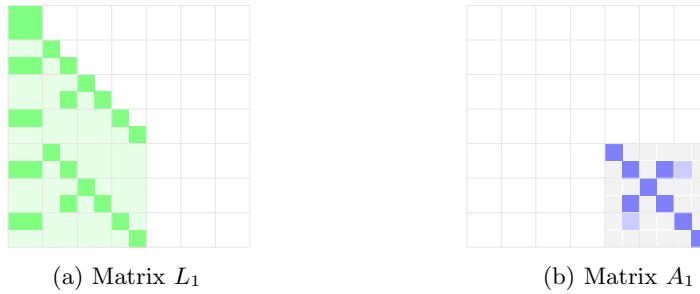


Fig. 2.6: Matrices L_1 and A_1 after the full sweep of compression-elimination procedure

We refer to the full sweep of the compression-elimination procedure described above as “one level” of elimination. Then we consider the matrix A_1 as a new block sparse matrix and again apply the compression-elimination procedure to it (“second level” of elimination).

2.4. Sparsity of the factors. Let us now study the sparsity of the factors L_1 and A_1 . First define the block sparsity pattern of a block sparse matrix. For matrix A with M block columns, M block rows and block size $B \times B$ define $\mathbf{bsp}(A)_{B \times B}^{M \times M}$ (block sparsity pattern of block-sparse matrix A) as a function

$$F_A(i, j) \rightarrow \{0, 1\}, \forall i, j = 1 \dots M$$

that returns 1 if the block A_{ij} is nonzero and 0 otherwise. For simplicity, assume that in the first level elimination at each step we eliminate $(B - r)$ rows. For the matrix \check{L}_1 , according to equations (2.5) and (2.6) we can see that

$$\mathbf{bsp}(\check{L}_1)_{B \times (B-r)}^{M \times M} = \mathbf{bsp}(A)_{B \times B}^{M \times M}.$$

The permutation matrix P_1 does not change the number of nonzero blocks in $L_1 = P_1 \check{L}_1$. Denote the number of nonzero blocks of matrix A as

$$\#\mathbf{bsp}(A)_{B \times B}^{M \times M}.$$

Since the permutation P_1 splits the blocks of the matrix L_1 into “eliminated” and “not eliminated” parts,

$$\#\mathbf{bsp}(L_1)_{(B-r) \times (B-r) \text{ or } r \times (B-r)}^{M \times M} = 2\#\mathbf{bsp}(\check{L}_1)_{B \times (B-r)}^{M \times M} = 2\#\mathbf{bsp}(A)_{B \times B}^{M \times M}. \quad (2.8)$$

Unlike the matrix L_1 , the sparsity of the matrix A_1 grows in a more complex way. Since the matrix A_1 is an initial point for the next level it is very important to estimate its sparsity. Thanks to (2.2) we can see that²

$$\mathbf{bsp}(A_1)_{r \times r}^{M \times M} \leq \mathbf{bsp}(A^2)_{B \times B}^{M \times M}.$$

The crucial part of the algorithm is that for the next level of elimination we combine the blocks of the matrix A_1 into super-blocks of size $rJ \times rJ$ ($Jr \approx B$). Then we consider the new matrix with M/J blocks (assume that M is divisible by J). The number J is the inner parameter of the algorithm. The new matrix is then considered as an $MJ \times MJ$ matrix. (The new block size is Jr .³) We can see that

$$\mathbf{bsp}(A_1)_{rJ \times Jr}^{(M/J) \times (M/J)} \leq \mathbf{bsp}(A^2)_{BJ \times BJ}^{(M/J) \times (M/J)}. \quad (2.9)$$

This is where the choice of the original ordering is required. We consider the choice of permutation in Section 2.7.

2.5. Multilevel step. Consider the matrix A_1 with $Jr \times Jr$ blocks and use the compression-elimination procedure to find corresponding matrices Q_2 , L_2 and P_2 :

$$P_2 Q_2 A_1 Q_2^\top P_2^\top \approx L_2 L_2^\top + \begin{bmatrix} 0_{n_{L_2} \times n_{L_2}} & 0 \\ 0 & A_2 \end{bmatrix}.$$

We introduce

$$\check{Q}_2 = \begin{bmatrix} I_{n_{L_1}} & 0 \\ 0 & Q_2 \end{bmatrix}, \quad \check{P}_2 = \begin{bmatrix} I_{n_{L_1}} & 0 \\ 0 & P_2 \end{bmatrix}, \quad \text{and} \quad \check{L}_2 = \begin{bmatrix} 0_{n_{L_1} \times n_{L_2}} \\ L_2 \end{bmatrix}.$$

²By $\mathbf{bsp}(A_1)_{r \times r}^{M \times M} \leq \mathbf{bsp}(A^2)_{B \times B}^{M \times M}$ we mean that $F_{A_1}(i, j) \leq F_{A^2}(i, j), \forall i, j = 1 \dots M$.

³We can easily adapt for variable block size, but for simplicity we consider them to be equal.

then we have two-level approximate factorization

$$\check{P}_2 \check{Q}_2 \check{P}_1 \check{Q}_1 A \check{Q}_1^\top \check{P}_1^\top \check{Q}_2^\top \check{P}_2^\top \approx \begin{bmatrix} L_1 & \check{L}_2 \end{bmatrix} \begin{bmatrix} L_1 & \check{L}_2 \end{bmatrix}^\top + \begin{bmatrix} 0 & 0 \\ 0 & A_2 \end{bmatrix}, \quad (2.10)$$

When the size is small we do the Cholesky factorization of the remainder. The algorithm is summarized in the next proposition.

PROPOSITION 2.1. *The compression-elimination (CE) algorithm leads to the approximate factorization*

$$A \approx Q^\top L L^\top Q, \quad (2.11)$$

where Q is a unitary matrix equal to the multiplication of block-diagonal and permutation matrices

$$Q = \prod_{j=1}^K \check{P}_j \check{Q}_j,$$

L is a sparse block lower-triangular matrix

$$L = \begin{bmatrix} \begin{bmatrix} L_1 \end{bmatrix} & \begin{bmatrix} 0 \\ L_2 \end{bmatrix} & \dots & \begin{bmatrix} 0 \\ \vdots \\ 0 \\ L_{K+1} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} L_1 & \check{L}_2 & \dots & \check{L}_{K+1} \end{bmatrix},$$

where block L_j has

$$\#L_j = 2\#\mathbf{bsp}(A_{j-1})_{rJ \times rJ}^{M/J^{j-1} \times M/J^{j-1}}$$

nonzero blocks, and the total number of nonzero blocks in factor L is

$$\begin{aligned} \#L = 2\#\mathbf{bsp}(A)_{B \times B}^{M \times M} + 2 \left(\sum_{j=1}^{K-1} \#\mathbf{bsp}(A^{2^j})_{BJ^j \times BJ^j}^{M/J^j \times M/J^j} \right) + \\ + \frac{1}{2}(M/J^K)^2 r^2. \end{aligned} \quad (2.12)$$

K is the number of levels in CE algorithm, B is the block size. Time complexity for the approximate CE factorization is $\mathcal{O}(B^3(\#L_{K+1} - \#L_1) + B^2\#L)$. Memory complexity is $\mathcal{O}(B(B-r)\#L + KNB)$. We call (2.11) the CE approximate factorization of the matrix A .

Proof. After K levels,

$$\check{P}_K \check{Q}_K \dots \check{P}_1 \check{Q}_1 A \check{Q}_1^\top \check{P}_1^\top \dots \check{Q}_K^\top \check{P}_K^\top \approx \begin{bmatrix} L_1 & \check{L}_2 & \dots & \check{L}_K \end{bmatrix} \begin{bmatrix} L_1^\top \\ \check{L}_2^\top \\ \vdots \\ \check{L}_K^\top \end{bmatrix} + \begin{bmatrix} 0_{n_* \times n_*} & 0 \\ 0 & A_K \end{bmatrix}, \quad (2.13)$$

where $n_* = \sum_{j=1}^K n_{L_j}$. Denote

$$Q = \prod_{j=1}^K \check{P}_j \check{Q}_j, \quad (2.14)$$

The remainder is factorized exactly as

$$A_K = L_{K+1} L_{K+1}^\top.$$

Denote

$$L = \begin{bmatrix} \begin{bmatrix} L_1 \end{bmatrix} & \begin{bmatrix} 0 \\ L_2 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ \vdots \\ 0 \\ L_{K+1} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} L_1 & \check{L}_2 & \cdots & \check{L}_{K+1} \end{bmatrix},$$

where L_j are rectangular block-triangular matrices. Using introduced notation and (2.13) we get (2.11). The number $\#L$ of nonzero blocks in the factor L is

$$\#L = \sum_{j=1}^{K+1} \#L_j. \quad (2.15)$$

Using (2.8),

$$\#L_1 = 2\#\mathbf{bsp}(A)_{B \times B}^{M/J \times M/J},$$

for $j = 2 \dots K$:

$$\#L_j = 2\#\mathbf{bsp}(A_{j-1})_{rJ \times rJ}^{M/J^{j-1} \times M/J^{j-1}}, \quad (2.16)$$

and for $j = K + 1$:

$$\#L_{K+1} = \frac{1}{2}(M/J^K)^2 r^2.$$

Using (2.9) we can see that

$$\begin{aligned} \mathbf{bsp}(A_K)_{r \times r}^{(M/J^K) \times (M/J^K)} &\leq \mathbf{bsp}(A_{K-1}^2)_{rJ \times rJ}^{(M/J^K) \times (M/J^K)} \leq \dots \\ &\dots \leq \mathbf{bsp}(A^{2^K})_{BJ^K \times BJ^K}^{(M/J^K) \times (M/J^K)}. \end{aligned}$$

Finally, we obtain equation (2.12).

In the CE algorithm we need to store Q and L factors. According to (2.14), the factor Q is a product of K permutation matrices $\check{P}_1, \dots, \check{P}_K$ and K block-diagonal matrices $\check{Q}_1, \dots, \check{Q}_K$ with block size B . Thus the matrix Q can be stored using $\mathcal{O}(KNB)$ memory. The memory requirement for the matrix L is $\mathcal{O}(B(B-r)\#L)$. Therefore, the factorization needs $\mathcal{O}(B(B-r)\#L + KNB)$ memory.

Now let us estimate the number of operations in the construction of the CE factorization. The CE factorization is done in K sweeps, each sweep consists of

several compression and elimination steps. Let us compute the computational cost of the i -th sweep. Compression step requires the computation of the SVD factorizations for the far blocks and multiplication of the block rows and columns by the left SVD factor. The matrix A_i has $(\#L_{i+2} - \#L_{i+1})$ far blocks of size $B \times B$, thus the SVD factorizations needs $\mathcal{O}(B^3(\#L_{i+2} - \#L_{i+1}))$ operations. The multiplication of the matrix A_i by the unitary block-diagonal matrix with block size B (since the matrix A_i has $\#L_{i+1}$ number of blocks) costs $\mathcal{O}(B^2\#L_{i+1})$ operations. Elimination procedure (using the block Cholesky factorization (2.1)) for all rows of the matrix A_i requires $\mathcal{O}((B-r)^2\#L_{i+1})$ operations. Thus, i -th sweep of the CE algorithm costs $\mathcal{O}((B-r)^3(\#L_{i+1} - \#L_i) + B^2\#L_{i+1})$. Total computational cost of K sweeps of the CE algorithm is

$$t = \sum_{i=0}^K (\mathcal{O}(B^3(\#L_{i+2} - \#L_{i+1}) + B^2\#L_{i+1})) =$$

$$\mathcal{O}(B^3(\#L_{K+1} - \#L_1)) + B^2 \sum_{i=0}^K \mathcal{O}(\#L_{i+1}),$$

taking into account (2.15),

$$t = \mathcal{O}(B^3(\#L_{K+1} - \#L_1) + B^2\#L).$$

□

PROPOSITION 2.2. *The system $Ax = b$ with a precomputed CE-factorization of the matrix A ($A = Q^\top LL^\top Q$) can be solved in $\mathcal{O}((B-r)B\#L + NKB)$ operations.*

Proof. Since

$$Q^\top LL^\top Qx = b,$$

then

$$x = Q^\top L^{-\top} L^{-1} Qb.$$

We need to compute the matrix Q and the matrix Q^\top by vector products. According to (2.14), the matrices Q and Q^\top are the product of K permutation matrices $\check{P}_1, \dots, \check{P}_K$ and K block-diagonal matrices $\check{Q}_1 \dots \check{Q}_K$ with block size B . The block-diagonal matrix (with block size B) by vector product requires $\mathcal{O}(NB)$ operations. Thus, the total computational cost of the matrix Q and the matrix Q^\top by vector products is $\mathcal{O}(NKB)$ operations. Since L is a block-triangular sparse matrix with $\#L$ nonzero $(B-r) \times B$ blocks, the solution of the system with matrix L requires $\mathcal{O}((B-r)B\#L)$ operations. Similarly, the solution of the system with matrix L^\top requires $\mathcal{O}((B-r)B\#L)$ operations. In total, we need $\mathcal{O}((B-r)B\#L + NKB)$ operations to solve the system $Q^\top LL^\top Qx = b$. □

2.6. Pseudo code. Here we present the scheme of the CE factorization algorithm.

Algorithm 1: CE algorithm

Input:
 $A \in \mathbb{R}^{n \times n} = \mathbb{R}^{MB \times MB}$ - block sparse matrix, $A_0 \equiv A$
 K - number of levels
 ε or r - rank revealing parameter

Factorization:
for $j = 1 \dots K$ **do**
 $M_j = \frac{n}{BJ(j-1)}$ - number of blocks on current level
Level elimination: (see the Algorithm 2)
Input : A_{j-1}, M_j, r or ε
Output : $\hat{A}_j, \check{L}_j, Q_j$
 $L_j = P_j^\top \check{L}_j$,
 $A_j = P_j \hat{A}_j P_j^\top$ - remainder.
Factorize $A_K = L_{K+1} L_{K+1}^\top$,
 $Q = \left(\prod_{j=1}^K P_j Q_j \right)$,
 $L = \begin{bmatrix} L_1 & \check{L}_2 & \dots & \check{L}_{K+1} \end{bmatrix}$.

Output:
 L, Q (where $A \approx Q^\top L L^\top Q$).

Algorithm 2: One level elimination

Level elimination

Input:
 A, M, ε or r

for $i = 1 \dots M$ **do**
Compression step:
 $\tilde{R}_i = \begin{bmatrix} D_i & C_i & F_i \end{bmatrix} P_i^{\text{col}}$ - i -th block row ,
 $F_i \approx \tilde{U}_i \begin{bmatrix} \hat{F}_i \\ 0 \end{bmatrix}$, \tilde{U}_i - unitary matrix, $\hat{F}_i \in \mathbb{R}^{r \times Bn_i^F}$,
 $\tilde{R}_{i*} = \begin{bmatrix} \tilde{U}_i^\top D_i \tilde{U}_i & \tilde{U}_i^\top C_i & \begin{bmatrix} \hat{F}_i \\ 0 \end{bmatrix} \end{bmatrix} P_i^{\text{col}}$.
Elimination step:
 $\tilde{S}_i = \begin{bmatrix} 0_{r \times r} & 0 \\ 0 & I_{(B-r)} \end{bmatrix} \tilde{R}_{i*}$,
Eliminate \tilde{S}_i using block Cholesky formula:
 $\hat{A}_{i-1} = \tilde{L}_i \tilde{L}_i^\top + \hat{A}_i$,
Add \tilde{L}_i into matrix \check{L} .
 $Q = \text{diag}(\tilde{U}_1, \dots, \tilde{U}_M)$.
Output : \hat{A}_M, \check{L}, Q

2.7. CE complexity based on the graph properties. Consider the undirected graph \mathcal{G}_A associated with the block sparsity pattern of the symmetric matrix A : i -th graph node corresponds to i -th block rows and columns, if the block (i, j) is nonzero then i -th and j -th graph nodes are connected by an edge. In this section we clarify the connection between properties of the graph \mathcal{G}_A and the complexity of the CE algorithm.

For example, consider the matrix A with the graph shown on Figure 2.7a. This matrix can arise from the 2D elliptic equation discretized on a uniform square grid in $[0, 1]^2$ by finite-difference scheme with 9-point stencil. Note that each graph node is connected only with a fixed number of nearest spatial neighbors (we call this property *the graph locality*.)

According to (2.9), one sweep of the CE algorithm leads to squaring of the block sparsity pattern of the remainder. Squaring of the matrix A spoils the locality of the graph \mathcal{G}_A : if two nodes were connected to the same node in the graph \mathcal{G}_A , they are connected in the graph \mathcal{G}_{A^2} (see the graph \mathcal{G}_{A^2} on Figure 2.7b). We call this step “squaring” ($\mathcal{G}_{A^2} = \mathbf{Sq}(\mathcal{G}_A)$). Note that the squaring procedure significantly increases the number of edges in the graph \mathcal{G}_{A^2} .

The next step of the CE factorization joins block rows and columns into super-blocks by groups of J . We obtain the matrix A_1 from equation (2.7). For the graph \mathcal{G}_{A^2} it means joining nodes into super-nodes by groups of J . We call this step “coarsening” $\mathcal{G}_{A_1} = \mathbf{Coars}(\mathcal{G}_{A^2})$. Coarsened graph \mathcal{G}_{A_1} has better locality than the graph \mathcal{G}_{A^2} .

In this particular example the graph structure of the initial matrix A and of the squared-coarsened matrix A_1 is very similar, see Figure. 2.7c. During the CE algorithm we obtain the graphs $\mathcal{G}_A, \mathcal{G}_{A_1}, \dots, \mathcal{G}_{A_K}$, where $\mathcal{G}_{A_i} = \mathbf{Coars}(\mathbf{Sq}(\mathcal{G}_{A_{i-1}}))$, $\forall i = 1, \dots, K$.

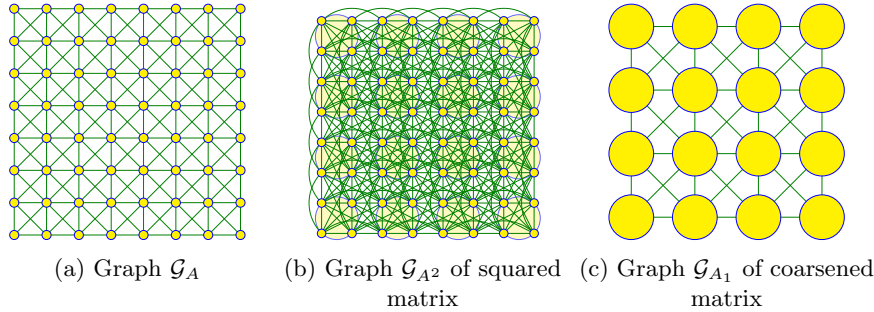


Fig. 2.7: Example of squaring-coarsening procedure for graph \mathcal{G}_A

Note that the number of edges in the graphs $\mathcal{G}_A, \mathcal{G}_{A_1}, \dots, \mathcal{G}_{A_K}$ is related to the number $\#L$, that we need to estimate, see Proposition 2.1.

PROPOSITION 2.3. *The number $\#L$ of nonzero blocks in the factor L is equal to*

the total number of edges of all graphs $\mathcal{G}_A, \mathcal{G}_{A_1}, \dots, \mathcal{G}_{A_K}$:

$$\#L = \mathbf{Edge_num}(\mathcal{G}_A) + \sum_{i=1}^K \mathbf{Edge_num}(\mathcal{G}_{A_i}).$$

Proof. The number of edges of the graph \mathcal{G}_{A_i} is equal to the number of nonzero blocks in the matrix $A_i \forall i \in 0 \dots K$ ($A_0 = A$) by the definition. Taking into account equation (2.12) we obtain:

$$\begin{aligned} \#L &= 2\#\mathbf{bsp}(A)_{B \times B}^{M \times M} + 2 \left(\sum_{j=1}^{K-1} \#\mathbf{bsp}(A^{2^j})_{B^{J^j} \times B^{J^j}}^{M/J^j \times M/J^j} \right) + \\ &+ \frac{1}{2}(M/J^K)^2 r^2 = \mathbf{Edge_num}(\mathcal{G}_A) + \sum_{i=1}^K \mathbf{Edge_num}(\mathcal{G}_{A_i}). \end{aligned} \quad (2.17)$$

□

Thus, the minimization of the number $\#L$ is equivalent to the minimization of the total number of edges in graphs $\mathcal{G}_A, \mathcal{G}_{A_1}, \dots, \mathcal{G}_{A_K}$.

The coarsening procedure is the key to the reduction of the number of edges. Note that the coarsening procedure for all graphs $\mathcal{G}_A, \mathcal{G}_{A_1}, \dots, \mathcal{G}_{A_K}$ can be defined by a single initial permutation P and the number of nodes J that we join together (neighbors in the permutation P are joined by the groups of J). Also note that, with known J , the coarsening procedure determines the initial permutation P . Thus, to minimize $\#L$, instead of optimizing the initial permutation P we can optimize the coarsening procedure, which can be much more intuitive.

Consider the example on Figure 2.7. The following proposition shows that for this example a good coarsening exists (and is very simple).

PROPOSITION 2.4. *Let the graph \mathcal{G}_A be defined by a tensor product grid in \mathbb{R}^2 , (grid points correspond to nodes of the graph \mathcal{G}_A , edges of the grid correspond to edges of the graph \mathcal{G}_A), like in the example on Figure 2.7. If the coarsening procedure joins the closest nodes and each super-node has at least two nodes in each direction, then each graph $\mathcal{G}_A, \mathcal{G}_{A_1}, \dots, \mathcal{G}_{A_K}$ has $\mathcal{O}(N)$ edges.*

Proof. Consider the graph \mathcal{G}_A based on the tensor product grid in \mathbb{R}^2 , let each node n of this graph have index (i, j) . Let $s(e)$ be the length of the edge e that connects nodes n_1 and n_2 with indices (i_1, j_1) and (i_2, j_2) if

$$s(e) = \max(|i_1 - i_2|, |j_1 - j_2|).$$

Let $\tilde{s}(\mathcal{G}_A)$ be the maximum edge length of the graph \mathcal{G}_A :

$$\tilde{s}(\mathcal{G}_A) = \max_{e_i \in \mathcal{G}_A} s(e_i).$$

For the graph \mathcal{G}_{A_i} of the matrix $A_i, \forall i \in 0 \dots (K-1)$, $\tilde{s}(\mathcal{G}_{A_i}) = 1$. By the squaring procedure for the graph $\mathcal{G}_{A_i^2} = \mathbf{Sq}(\mathcal{G}_{A_i})$, $\tilde{s}(\mathcal{G}_{A_i^2}) = 2$. Let us prove that after the coarsening procedure described in the proposition statement ($\mathcal{G}_{A_{i+1}} = \mathbf{Coars}(\mathcal{G}_{A_i^2})$) we have $\tilde{s}(\mathcal{G}_{A_{i+1}}) = 1$. If $\tilde{s}(\mathcal{G}_{A_{i+1}}) > 1$, then, since each super-node has at least two nodes in each direction, $\tilde{s}(\mathcal{G}_{A_i^2}) > 3$, this is a contradiction. Thus, $\tilde{s}(\mathcal{G}_{A_{i+1}}) = 1$.

If $\tilde{s}(\mathcal{G}_{A_{i+1}}) = 1$, then each node has a constant number of edges. Thus, the number of edges in the graph $\mathcal{G}_{A_i}, \forall i \in 0 \dots (K-1)$ is $\mathcal{O}(N)$. □

COROLLARY 2.5. *Proposition 2.4 is also true for $\mathbb{R}^d, d > 2$.*

Proof. Analogously to Proposition 2.3. \square

COROLLARY 2.6. *Let the matrix A_{tpg} have the graph based on the tensor product grid in \mathbb{R}^d . The CE factorization of the matrix A_{tpg} requires $\mathcal{O}(B^3N + B^2NK)$ operations and $\mathcal{O}(B(B-r)NK)$ memory.*

Proof. According to Proposition 2.4, each graph $\mathcal{G}_A, \mathcal{G}_{A_1}, \dots, \mathcal{G}_{A_K}$ has $\mathcal{O}(N)$ edges. Thus, by Proposition 2.3, $\#L = \mathcal{O}(NK)$, $(\#L_{K+1} - \#L_1) = \mathcal{O}(N)$. Therefore, by Proposition 2.1, memory complexity of the CE factorization of the matrix A_{tpg} is the following:

$$\text{mem} = \mathcal{O}(B(B-r)\#L + KNB) = \mathcal{O}(B(B-r)NK + KNB) = \mathcal{O}(B(B-r)NK),$$

and the time complexity:

$$\mathbf{t} = \mathcal{O}(B^3(\#L_{K+1} - \#L_1) + B^2\#L) = \mathcal{O}(B^3N + B^2NK).$$

\square

Note, that graph partitioning algorithms [29, 14] and, in particular, the nested dissection procedure [15] can be used for grouping the closest blocks during the coarsening procedure.

Let us now discuss the choice of the coarsening procedure in the case of geometric graphs. Consider the k -nearest neighborhood graph [26]. This graph has a geometric interpretation, where each node has at most k edges and has a sphere that contains all connected nodes and only them. For example, the graph \mathcal{G}_A on Figure 2.7b is a 8-nearest neighborhood graph. We propose the following hypothesis.

HYPOTHESIS 1. *Let graph \mathcal{G}_A be a k -nearest neighborhood graph; let h_{\min} and h_{\max} be the minimum and maximum edge lengths; let B be the maximum block size. Let graphs $\mathcal{G}_{A_1}, \dots, \mathcal{G}_{A_K}$ be obtained in K squaring-coarsening steps. Then there exists a coarsening procedure and a number*

$$k_1 = k_1 \left(\frac{h_{\max}}{h_{\min}}, d, B \right),$$

such that the graphs $\mathcal{G}_{A_1}, \dots, \mathcal{G}_{A_K}$ are k_1 -nearest neighborhood graphs.

REMARK 2.2. *The k -nearest neighborhood graph has $\mathcal{O}(N)$ edges if k does not depend on N . Thus, Hypothesis 1 can be reformulated in the following way: if \mathcal{G}_A is a k -nearest neighborhood graph, then there exists a coarsening procedure such that each of the graphs $\mathcal{G}_{A_1}, \dots, \mathcal{G}_{A_K}$ has $\mathcal{O}(N)$ edges. Thus, analogously to Corollary 2.6, the CE factorization of the matrix with k -nearest neighborhood graph requires $\mathcal{O}(B^3N + B^2NK)$ operations and $\mathcal{O}(B(Br)NK)$ memory.*

3. Numerical experiments. We compare the following solvers: $\text{CE}(\varepsilon)$ with backward step as a direct solver, $\text{CE}(\varepsilon)$ and $\text{CE}(r)$ factorizations as preconditioners for iterative symmetric solver MINRES, direct symmetric solver from the package CHOLMOD and MINRES with ILUt preconditioner. The required accuracy is set to 10^{-10} . Note that for the solution of PDEs this accuracy is much less than the discretization accuracy, so often a much larger threshold is sufficient.

For the tests we consider the system obtained from an equidistant cubic discretization of a 3D diffusion equation.

$$\begin{aligned} \operatorname{div} (k(x) \operatorname{grad} u(x)) &= f(x), \quad x \in \Omega, \\ u|_{\partial\Omega} &= 0 \end{aligned} \tag{3.1}$$

where $\Omega = [0, 1]^3$, the diffusion tensor $k(x) = \text{diag}(x_1^2 + 0.5, x_2^2 + 0.5, x_3^2 + 0.5)$, right hand side $f(x) = 1$. CE algorithm is implemented in Fortran using BLAS from Intel MKL. For MINRES we use Python library SciPy. Computations were performed on a server with 32 Intel® Xeon® E5-2640 v2 (20M Cache, 2.00 GHz) processors and with 256GB of RAM. Tests were run in a single-processor mode.

3.1. $\text{CE}(\varepsilon)$ factorization as a direct solver. The factorization obtained in $\text{CE}(r)$ procedure is not accurate enough to use it as a direct solver. Consider $\text{CE}(\varepsilon)$ factorization as an approximate direct solver. In Table 3.1 we show the relative accuracy η that can be achieved by a direct solution of the system with a matrix approximated by $\text{CE}(\varepsilon)$ factorization.

Revealing parameter, ε	10^{-2}	10^{-4}	10^{-6}	10^{-8}
Relative accuracy, η	4.0×10^{-1}	9.1×10^{-3}	1.2×10^{-5}	9.9×10^{-7}
Required memory, MB	553	1348	2494	2671
Time, sec	3.40107	8.28388	17.40455	29.65910

Table 3.1: Factorization accuracy and required memory, $N = 65536$. Accuracy η here is computed as: $\eta = \frac{\|x_{CE} - x_*\|}{\|x_*\|}$, where x_{CE} is the obtained the solution and x_* is the exact solution.

The main problem with the direct $\text{CE}(\varepsilon)$ solver is that it requires too much memory. On the other hand, $\text{CE}(\varepsilon)$ with large enough ε works fast and requires small amount of memory. This factorization as well as the $\text{CE}(r)$ factorization can be utilized as a very good preconditioner for iterative solvers, e.g. MINRES.

3.2. $\text{CE}(\varepsilon)$, $\text{CE}(r)$ convergence. On Figure 3.1 the convergence of MINRES with $\text{CE}(\varepsilon)$, $\text{CE}(r)$, ILUT preconditioners and MINRES without preconditioner are shown.

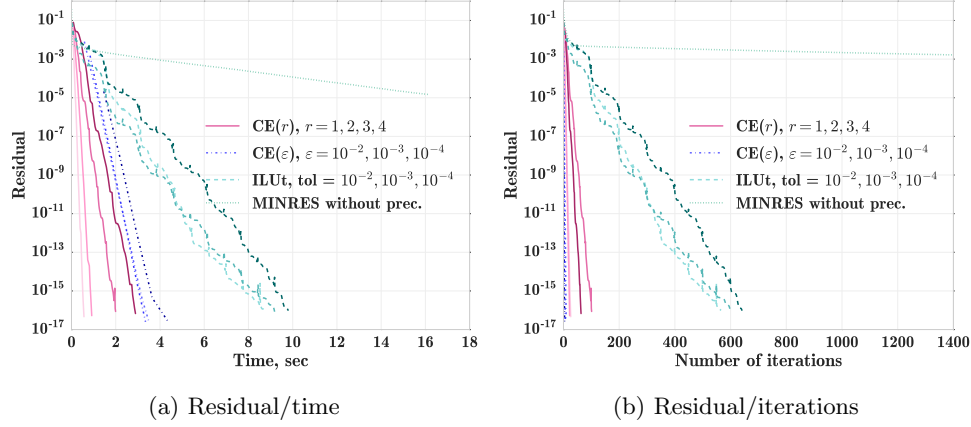


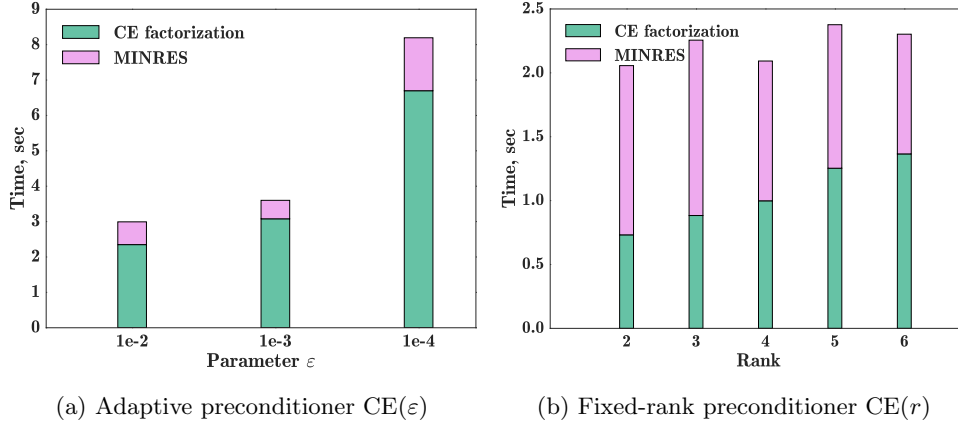
Fig. 3.1: Convergence comparison of MINRES with different preconditioners, $N = 32768$. Note that the line style and the line color on this figure mark the type of the preconditioner, different parameters have different color tones, from dark to light in the list of parameters.

REMARK 3.1. Note that the number of MINRES iterations without the preconditioner as well as the number of MINRES iterations with the $ILUt$ preconditioner grows as the mesh size increases, because the matrix becomes increasingly ill-conditioned. $CE(\varepsilon)$ and $CE(r)$ are much better preconditioners. For $CE(\varepsilon)$ preconditioner with fixed accuracy number of iterations does not grow; for $CE(r)$ preconditioner growth of the number of iterations is very mild (Table 3.2).

Matrix size, N	8192	16384	32768	65536	131072
Without preconditioner	147	2636	> 1000	> 1000	> 1000
$ILUt$, $t = 10^{-3}$	40	92	339	> 1000	> 1000
$CE(r)$, $r = 4$	23	25	29	30	36
$CE(\varepsilon)$, $\varepsilon = 10^{-3}$	4	5	6	5	6

Table 3.2: The number of iterations required for MINRES to converge to accuracy $\epsilon = 10^{-10}$ for different preconditioners.

We have the standard trade off: the smaller the ε is, the better the convergence is, but the storage grows. This fact is illustrated on Figure 3.2.

Fig. 3.2: Factorization and MINRES time, $N = 32768$

Note that on Figure 3.2b the total time for different ranks is the same, but for $r = 2$ less memory is required. Also note that $CE(\varepsilon)$ and $CE(r)$ have comparable timings so let us compare their memory requirements for different N .

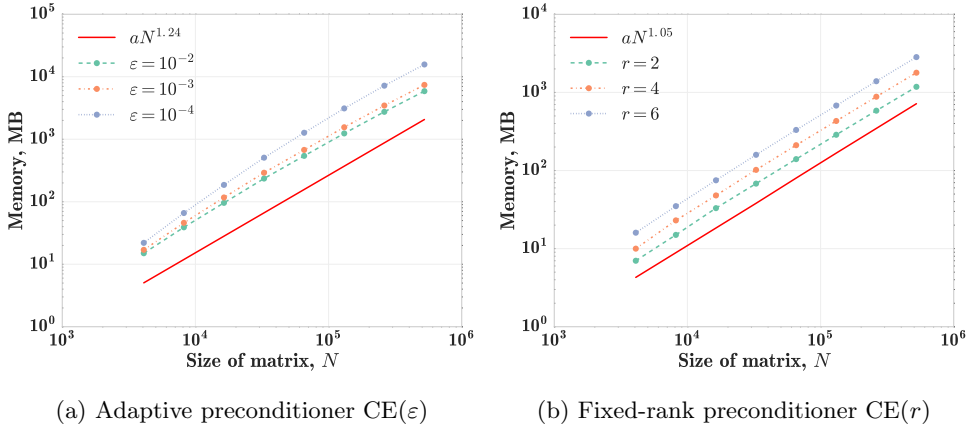


Fig. 3.3: Memory requirements for approximate factorization

The memory requirements for the fixed-rank preconditioner are predictably lower than for the adaptive ones. On Figure 3.4 the total time required for $CE(\varepsilon)$ and $CE(r)$ preconditioners and MINRES are shown for different N .

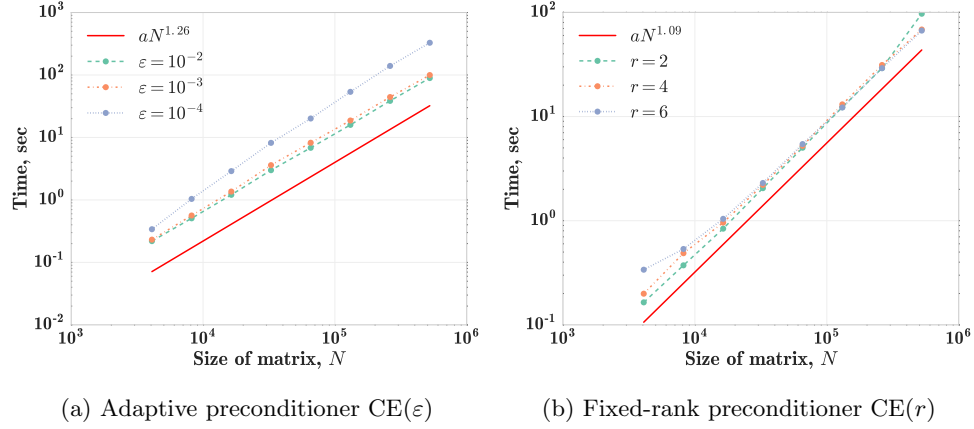


Fig. 3.4: Total solution time for MINRES with $CE(\varepsilon)$ and $CE(r)$ preconditioners

We have found experimentally that for the $CE(r)$ preconditioner $r = 4$ is typically a good choice, note that in this case $r = \frac{B}{2}$. For the $CE(\varepsilon)$ preconditioner a good choice is $\varepsilon = 10^{-2}$ (for this particular example).

3.2.1. Final comparison. Let us now compare $CE(\varepsilon)$ solver, $CE(r)$ solver, CHOLMOD and MINRES with ILUt preconditioner. Since MINRES with ILUt does not converge in 1000 iterations for $N > 32768$, we show its performance only for the points where it converged.

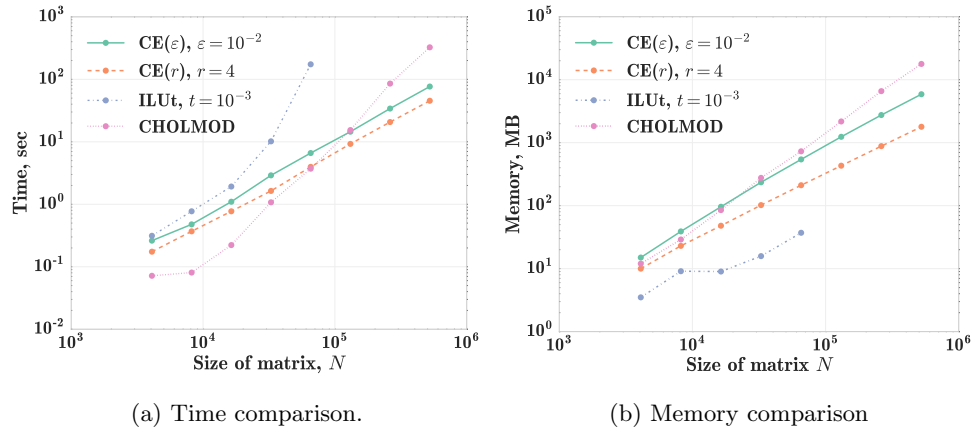


Fig. 3.5: Final solvers comparison (note logarithmic scales).

Note that CE(r) solver starts to use less memory than CHOLMOD at $N \approx 5000$ and starts to work faster at $N \approx 10000$. For the largest N our hardware permits we are able to solve the system ~ 10 times faster than CHOLMOD, and require ~ 10 times less memory. Note that the accuracy of CHOLMOD solution is 10^{-11} .

4. Related work. There are two big directions in fast solvers for sparse linear systems: sparse (approximate) factorizations and low-rank based solvers. CE-algorithm has the flavor of both.

The main difference between CE-algorithm and the classical direct sparse LU-like solvers (e.g. block Gaussian elimination [5], multifrontal method [4] and others [13, 27, 11, 8]) is that in our algorithm fill-in growth is controlled while maintaining the accuracy, thanks to the additional compression procedure. This leads to big advantage in memory usage and probably makes CE-algorithm asymptotically faster, but this point requires additional analysis. Note that references [4] and [8] use compressed representations (i.e. they are not just using the graph structure of the matrix, as is the case for the other solvers).

CE-algorithm is close to *hierarchical solvers*. These solvers utilize the *hierarchical structure* of matrices A and/or inverse matrix A^{-1} . For example \mathcal{H} direct solvers [17, 24, 7, 6, 16] build the LU factorization and/or the approximation of the inverse A^{-1} in the \mathcal{H} and \mathcal{H}^2 formats. Another example of hierarchical inversion methods is divide and conquer sparse preconditioning [22]. This type of solvers has provable linear complexity for matrices coming as discretization of PDEs, but the constant hidden in $\mathcal{O}(N)$ can be quite large, making them non-competitive. Hierarchical inversion methods (\mathcal{H} -LU method, divide and conquer sparse preconditioning, etc.) and CE solver are based on similar high-level ideas, but the algorithms are different: hierarchical inversion solvers utilize recursion, while CE algorithm is going from smallest to largest blocks.

Recently, so-called *HSS-solvers* have attracted a lot of attention. To name few references: [9, 10, 18, 30, 25]. As a subclass of such solvers, *HODLR* direct solvers [1, 3, 21] have been introduced. These solvers compute approximate sparse LU-like factorization of HSS-matrices. The simplicity of the structure in comparison to the general \mathcal{H}^2 structure allows for a very efficient implementation, and despite the fact that these matrices are essentially 1D- \mathcal{H}^2 matrices and optimal linear complexity is not possible for the matrices that are discretization of 2D/3D PDEs the running times can be quite impressive.

This work is closely related to the work by Ying and Ho with skeletonization: they use similar idea but compress all off-diagonal blocks, while we compress only the far away ones [20, 19, 23].

The \mathcal{H} -matrices have close connection to the fast multipole method (FMM), and an *inverse FMM solver* [2, 12] aimed at the inversion of integral transforms given by the FMM-structure. This method can be also adapted to the sparse matrix case as in paper [28]. The CE-algorithm has a simpler logic and structure, similar to the standard LU-factorization techniques accompanied by the compression procedure.

5. Conclusions and future work. We have proposed a new approximate factorization for sparse SPD matrices which provides better approximation to the matrix than standard incomplete LU factorization techniques. In our model experiments we were able to outperform CHOLMOD software in terms of memory cost and computational time. There are several directions for future research. At the moment, the permutation is assumed to be known by the start of the algorithm. In many cases it can be retrieved from the geometric information. However, it would be interesting

to develop the version that computes the next row to be eliminated in the adaptive way. Another important topic is to extend the solver for the non-symmetric case and also for dense \mathcal{H}^2 matrices. It is quite straightforward, but a lot of questions are still open. Finally, the efficient implementation of CE requires many small optimization of the original prototype code: in the current version most of the time is spent in the SVD procedure in the compression step, and using a cheaper rank-revealing factorization may significantly reduce the computational time. In the end, a much more broad comparison with other available solvers is needed to see the limitations of our approach in practice, as well as the theoretical justification of the low-rank property of the far blocks that appear in the process.

Acknowledgments. The authors would like to thank Maxim Rakhuba, Igor Ostanin, Alexander Novikov, Alexander Katrutsa and Marina Munkhoeva for their help for improving the paper.

REFERENCES

- [1] S. AMBIKASARAN AND E. DARVE, *An $\mathcal{O}(n \log n)$ fast direct solver for partial hierarchically semi-separable matrices*, J. Sci. Comput., 57 (2013), pp. 477–501.
- [2] S. AMBIKASARAN AND E. DARVE, *The inverse fast multipole method*, arXiv preprint arXiv:1309.1773, (2014).
- [3] A. AMINFAR, S. AMBIKASARAN, AND E. DARVE, *A fast block low-rank dense solver with applications to finite-element matrices*, J. Comput. Phys., 304 (2016), pp. 170–188.
- [4] A. AMINFAR AND E. DARVE, *A fast, memory efficient and robust sparse preconditioner based on a multifrontal approach with applications to finite-element matrices*, Int. J. Numer. Meth. Eng., (2015).
- [5] R. E. BANK, *Marching algorithms and block gaussian elimination*, Sparse Matrix Computations, (1976), pp. 293–307.
- [6] M. BEBENDORF, *Hierarchical LU decomposition-based preconditioners for BEM*, Computing, 74 (2005), pp. 225–247.
- [7] M. BEBENDORF AND W. HACKBUSCH, *Existence of \mathcal{H} -matrix approximants to the inverse FE-matrix of elliptic operators with L^∞ -coefficients*, Numer. Math., 95 (2003), pp. 1–28.
- [8] J. N. CHADWICK AND D. S. BINDEL, *An efficient solver for sparse linear systems based on rank-structured cholesky factorization*, arXiv preprint arXiv:1507.05593, (2015).
- [9] S. CHANDRASEKARAN, P. DEWILDE, M. GU, W. LYONS, AND T. PALS, *A fast solver for HSS representations via sparse matrices*, SIAM J. Matrix Anal. A., 29 (2006), pp. 67–81.
- [10] S. CHANDRASEKARAN, M. GU, AND T. PALS, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. A., 28 (2006), pp. 603–622.
- [11] Y. CHEN, T. A. DAVIS, W. W. HAGER, AND S. RAJAMANICKAM, *Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate*, ACM T. Math. Software, 35 (2008), p. 22.
- [12] P. COULIER, H. POURANSARI, AND E. DARVE, *The inverse fast multipole method: using a fast approximate direct solver as a preconditioner for dense linear systems*, arXiv preprint arXiv:1508.01835, (2015).
- [13] T. A. DAVIS, *Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method*, ACM T. Math. Software, 30 (2004), pp. 196–199.
- [14] K. D. DEVINE, E. G. BOMAN, R. T. HEAPHY, R. H. BISSELING, AND U. V. CATALYUREK, *Parallel hypergraph partitioning for scientific computing*, in Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, IEEE, 2006, pp. 10–pp.
- [15] A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.
- [16] L. GRASEDYCK, W. HACKBUSCH, AND R. KRIEMANN, *Performance of \mathcal{H} – lu preconditioning for sparse matrices*, Comput. Methods Appl. Math., 8 (2008), pp. 336–349.
- [17] W. HACKBUSCH, *Hlib package*. <http://www.hlib.org/>.
- [18] K. HO AND L. GREENGARD, *A fast direct solver for structured linear systems by recursive skeletonization*, SIAM J. Sci. Comput., 34 (2012), pp. A2507–A2532.
- [19] K. L. HO AND L. YING, *Hierarchical interpolative factorization for elliptic operators: differential equations*, Communications on Pure and Applied Mathematics, (2015).

- [20] ———, *Hierarchical interpolative factorization for elliptic operators: integral equations*, Communications on Pure and Applied Mathematics, (2015).
- [21] W. Y. KONG, J. BREMER, AND V. ROKHLIN, *An adaptive fast direct solver for boundary integral equations in two dimensions*, Appl. Comput. Harmon. A., 31 (2011), pp. 346–369.
- [22] R. LI AND Y. SAAD, *Divide and conquer low-rank preconditioning techniques*, tech. rep., Technical Report ys-2012-3. Dept. Computer Science and Engineering, University of Minnesota, Minneapolis, 2012.
- [23] Y. LI AND L. YING, *Distributed-memory hierarchical interpolative factorization*, arXiv preprint arXiv:1607.00346, (2016).
- [24] P.-G. MARTINSSON, *A fast direct solver for a class of elliptic partial differential equations*, J. Sci. Comput., 38 (2009), pp. 316–330.
- [25] P.-G. MARTINSSON AND V. ROKHLIN, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys., 205 (2005), pp. 1–23.
- [26] G. L. MILLER, S.-H. TENG, AND S. A. VAVASIS, *A unified geometric approach to graph separators*, in Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on, IEEE, 1991, pp. 538–547.
- [27] E. NG AND B. W. PEYTON, *A supernodal cholesky factorization algorithm for shared-memory multiprocessors*, SIAM J. Sci. Comput., 14 (1993), pp. 761–769.
- [28] H. POURANSARI, P. COULIER, AND E. DARVE, *Fast hierarchical solvers for sparse matrices*, arXiv preprint arXiv:1510.07363, (2015).
- [29] S. RAJAMANICKAM AND E. G. BOMAN, *Parallel partitioning with zoltan: Is hypergraph partitioning worth it?*, Contemp. Math., 588 (2012), pp. 37–52.
- [30] F.-H. ROUET, X. S. LI, P. GHYSELS, AND A. NAPOV, *A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization*, arXiv preprint arXiv:1503.05464, (2015).
- [31] Y. SAAD, *ILUT: A dual threshold incomplete LU factorization*, Numer. Linear Algebr., 1 (1994), pp. 387–402.